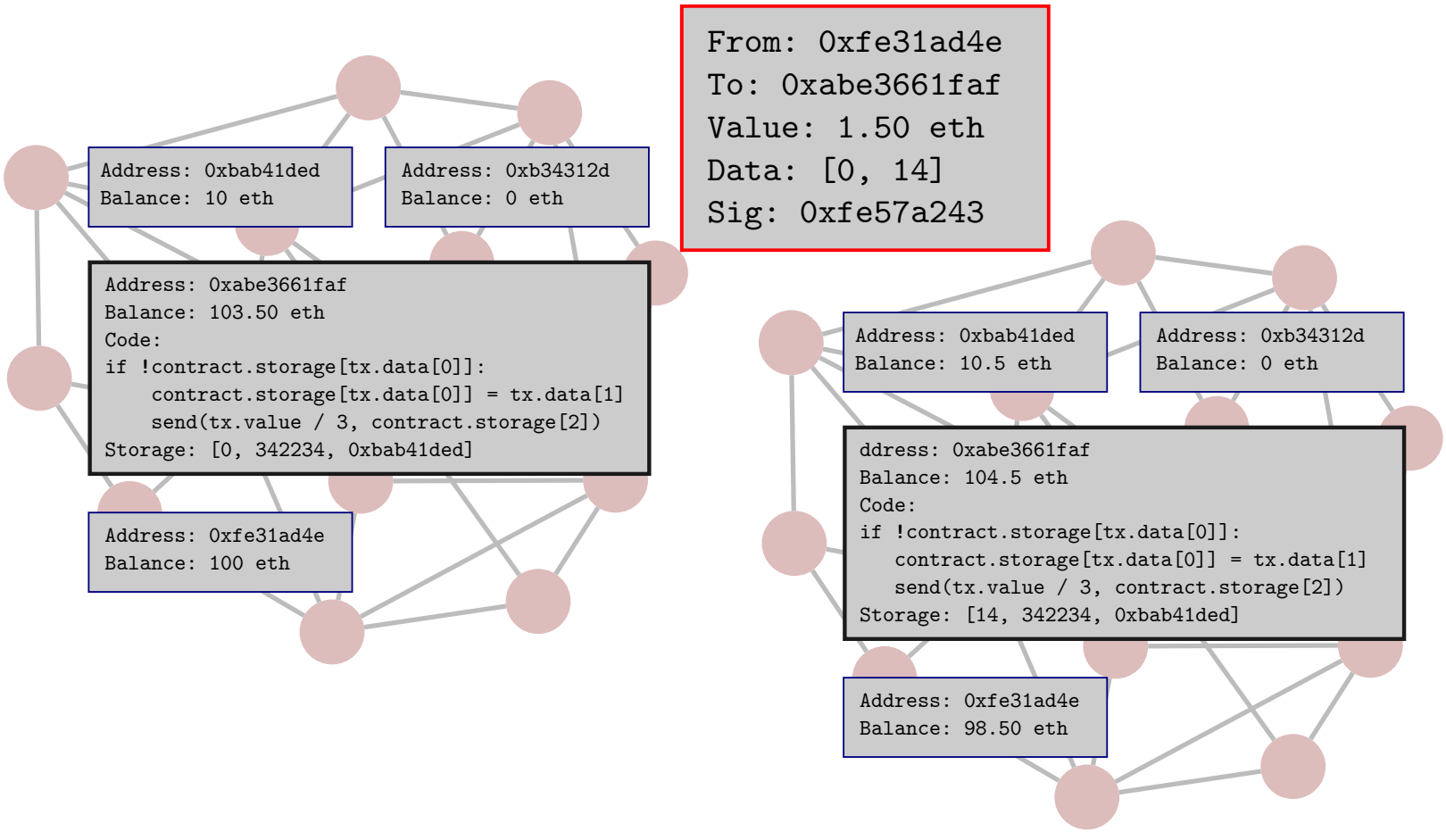
Raspodijeljene glavne knjige i kriptovalute

Jezik Solidity, Ethereum protokol

Ante Đerek, Zvonko Konstanjčar

20. prosinca 2021.

Ponavljanje: Pametni ugovori



Kako postići raspodijeljeni konsenzus?

- Niz transakcija zajedno s krajnjim globalnim stanjem čini *blok*.
- Distribuirani konsenzus slično kao kod "običnih" kriptovaluta (*proof-of-work*).

Tko izvršava ugovore?

- Rudari izvršavaju pozive ugovora prilikom slaganja bloka.
- Svi čvorovi izvršavaju pozive ugovora prilikom prihvaćanja bloka.

Posljedice raspodijeljenog konzenzusa

- Svi čvorovi moraju izvršavati kod i provjeriti rezultat.
- Izvršavanje mora biti deterministično.
- Svaki poziv se mora izvršiti do kraja ili se ne izvršiti uopće.

Izražajnost ugovora (najčešće) nije ograničena

Prilikom izvršavanja ugovor može:

- računati bilo što (jezik je *Turing potpun*),
- slati kriptovalutu,
- pozivati druge ugovore.

Ugovor se izvršava u *ograničenom* kontekstu

Ako transakcija T poziva ugovor C, ugovoru su dostupni samo:

- trajna memorija ugovora C,
- podaci iz transakcije T,
- izvor transakcije T (autentificiran digitalnim potpisom),
- metapodaci iz trenutnog i nedavnih blokova,
- rezultati eventualnih poziva drugih ugovora od strane C-a.

Gdje su poticaji za izvršavanje ugovora!

- Zašto bi rudar trošio resurse za izvršavanje ugovora kad može samo obrađivati *obične* transakcije?
- Što ako se poziv ugovora dugo (ili beskonačno) izvršava? Što ako troši nerazumnu količinu memorije?

Rješenje: gorivo

Inicijator transakcije plaća resurse potrebne za izvršavanje *gorivom*. U svakoj transakciji inicijator navodi:

- Gornji limit na količinu goriva predviđenog za izvršavanje transakcije.
- Cijenu naknade koju je voljan platiti po jedinici goriva.

```
contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    constructor() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
    }
}
```

Izvor: `solidity.readthedocs.io`

Pazi!

Baratanje s pogreškama je važan dio logike ugovora i česti izvor skupih grešaka.

Ako izvršavanje ugovora rezultira exceptionom onda transakcija ne mijenja globalno stanje. Funkcije: `assert`, `require`, `revert`.

```
contract Sharer {
    function sendHalf(address payable addr) public payable
      returns (uint balance) {
        require(msg.value % 2 == 0, "Even value required.");
        uint balanceBeforeTransfer = address(this).balance;
        addr.transfer(msg.value / 2);
        assert(address(this).balance ==
            balanceBeforeTransfer - msg.value / 2);
        return address(this).balance;
    }
}
```

Izvor: `solidity.readthedocs.io`

Pazi!

Bitno je znati što rade funkcije koje zovete u slučaju pogreške.

```
addr.transfer(funds / 2);
funds = funds - funds / 2;
```

```
addr.send(funds / 2);
funds = funds - funds / 2;
```

Pazi!

Bitno je pratiti novosti :)

Don't use transfer() or send() ...

Izvor: `consensys.github.io/smart-contract-best-practices`

Stvaranje novog ugovora i interakcija s njim

```
import "Faucet.sol";

contract Token {
    Faucet _faucet;

    constructor() {
        _faucet = (new Faucet).value(0.5 ether)();
    }

    function destroy() ownerOnly {
        _faucet.destroy();
    }
}
```

Izvor: github.com/ethereumbook

Interakcija s postojećim ugovorom

```
import "Faucet.sol";

contract Token is mortal {
    Faucet _faucet;
    constructor(address _f) {
        _faucet = Faucet(_f);
        _faucet.withdraw(0.1 ether)
    }
}
```

Izvor: `github.com/ethereumbook`

Pazi!

Svaka interakcija s drugim ugovorom je opasna.

- Je li poznat uopće kod drugog ugovora?
- Što ako pozvani ugovor opet pozove originalni ugovor?
- ...

Nismo pričali o puno toga:

- Destruktori.
- `private`, `public`, `payable`, ...
- *Function modifiers*.
- Nasljeđivanje ugovora.
- Eventi.
- Razvoj i testiranje.
- Kako procijeniti potrebno gorivo.
- Sigurnosni propusti i dobra praksa pisanja ugovora.
- Dapps.
- Oracles.
- ...

Primjene pametnih ugovora – podvalute

Ethereum token

Kriptovaluta implementirana kao pametni ugovor na Ethereum platformi.

ERC20

Tehnički standard za implementaciju tokena.

Primjeri

EOS, Filecoin, BNB, ...

```
...
function name() public view returns (string)
function symbol() public view returns (string)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
...
function transfer(address _to, uint256 _value)
    public returns (bool success)
function approve(address _spender, uint256 _value)
    public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value)
    public returns (bool success)
...
```

Decentralizirane burze

Pametni ugovori koji omogućuju razmjenu i trgovanje tokenima.

Krediti i ulaganje

Pametni ugovori koji omogućuju ulaganje tokena te uzimanje zajmova uz kamatu.

Primjeri

Uniswap, IDEX, Aave

Primjene pametnih ugovora – NFT

NFT

Omogućavaju bilježenje "vlasništva" i razmjenu digitalnih sadržaja.

Primjeri

OpenSea, Upland

Sustavi za registraciju imena

Pametni ugovor održava bazu koja imenima pridružuje adrese.

Reputacijski sustavi

Pametni ugovor prikuplja certifikate na kojima se zasniva reputacija i povjerenje.

Primjeri

`https://ens.domains/`

Sustavi za registraciju imena

Omogućavaju sučelje sa “stvarnim svijetom”.

Primjeri

ChainLink

Raspodijeljena autonomna organizacija

Virtualni entitet, suvlasnici glasanjem kolektivno odlučuju na koji način djelovati, potrošiti, probati zaraditi novce.

Jednostavan primjer

- Skup pametnih ugovora čije se ponašanje može mijenjati i proširivati (npr. kreiranjem novih ugovora.)
- Svaki suvlasnik može predložiti novu funkcionalnost (npr. u obliku ugovora).
  - Suvlasnici glasaju o prijedlogu.
  - Ako više od dvije trećine suvlasnika podrži prijedlog, on se automatski izvršava.

Neslavni primjer:

"The DAO".

Puno potencijalnih primjena ...

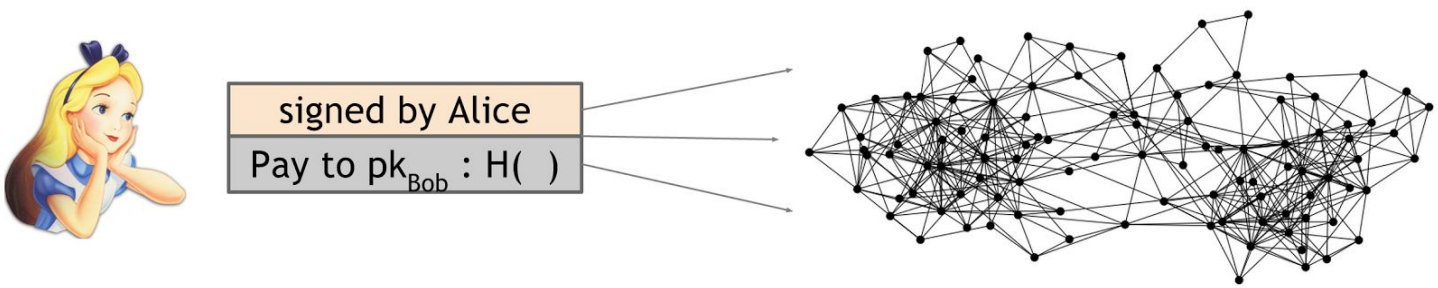
- Financijski derivati.
- Aukcije.
- Glasanje.
- Kockanje i klađenje.
- `https://www.stateofthedapps.com/`

... i puno problema.

- Zašto je uopće potrebno koristiti pametni ugovor?
- Kako se uvjeriti da je pametni ugovor ispravan?
- Kako na dobar način povezati ugovore sa stvarnim svijetom?
- ...

Arhitektura sustava

- Puno čvorova u "peer-to-peer" mreži.
- Svi čvorovi imaju skoro identične kopije lanca blokova.
- Svaki čvor održava skup transakcija koje treba dodati u lanac.
- Periodički se dodaje novi blok u lanac:
  - Čvor koji riješi kriptografsku slagalicu predloži sljedeći blok.
  - Svaki čvor provjeri ispravnost bloka prije prihvaćanja.



Izvor: `bitcoinbook.cs.princeton.edu`

Kriptografski primitivi u Ethereumu

- Hash funkcija: Keccak256
- Digitalni potpis: ECDSA nad secp256k1 (kao i Bitcoin)

Adresa

- EOA: zadnjih 20 byte-ova sažetka javnog ključa.
- Ugovor: zadnjih 20 byte-ova sažetka para (A, n_A) gdje je A adresa stvaratelja, a n_A redni broj transakcije s adrese A koja je stvorila ugovor.

Fakultet elektrotehnike i računarstva

Unutar lanca blokova i prilikom komunikacije na Ethereum mreži:

- Transakcija je zapisana u kompaktnom binarnom obliku.
- Hash transakcije služi kao njezin identifikator.

```
{
 nonce: 2,
 gasPrice: 0.0000000765 ether,
 gasLimit: 21000,
 to: '0x0975ca9f986eee35f5cbba2d672ad9bc8d2a0844',
 value: 0.02893164 ether,
 data: '',
 v: 38,
 r: '0xd2b0d401b543872d2a6a50de92455decbb868440321bf63a13b310c069e2ba5b',
 s: '0x3c6d51bcb2e1653be86546b87f8a12ddb45b6d4e568420299b96f64c19701040'
}
```

Bitcoin (UXTO)

Stanje računa je skup svih njegovih nepotrošenih novčića.

Ethereum (Account)

Stanje računa je broj — količina kriptovalute koju posjeduje.

Koje su razlike između dva pristupa?

- Gdje je veća doza privatnosti?
- Što je jednostavnije implementirati?
- Što bolje skalira?
- Što troši više prostora?

Uloge

- Štiti od napada ponovnim slanjem iste transakcije.
- Omogućava korisniku da poreda istovremene transakcije.

Implementacija

- Globalno stanje za svaku adresu A uključuje i broj transakcija n_A kojima je ta adresa bila izvor.
- Transakcija s adrese A je ispravna (u trenutnom globalnom stanju) samo ako je njezin *nonce* jednak n_A.

Razlika između "obične" transakcije i poziva ugovora?

```
{
 nonce: 476,
 gasPrice: 0.00000008 Ether,
 gasLimit: 210000,
 to: '0x9041fe5b3fdea0f5e4afdc17e75180738d877a01',
 value: 0 Ether,
 data: '0xa9059cbb000000000000000000000051093b3e69a6ab781ad596866...',
 v: 37,
 r: '0x9e8915ec764e62c3903618742cb8fc5b2daeebbfd0f705d978c8e84ba3074f31',
 s: '0x649961191caa11a0f787dbe1597464b04d73fc9cf007eb927b3d48fa5ba5bdb5'
}
```

Stvaranje ugovora?

```
...
 to: ''
 data: '0x6060604052604080519081016040526006808252f50524f312e300000...'
...
```

Globalnos stanje (*World State*)

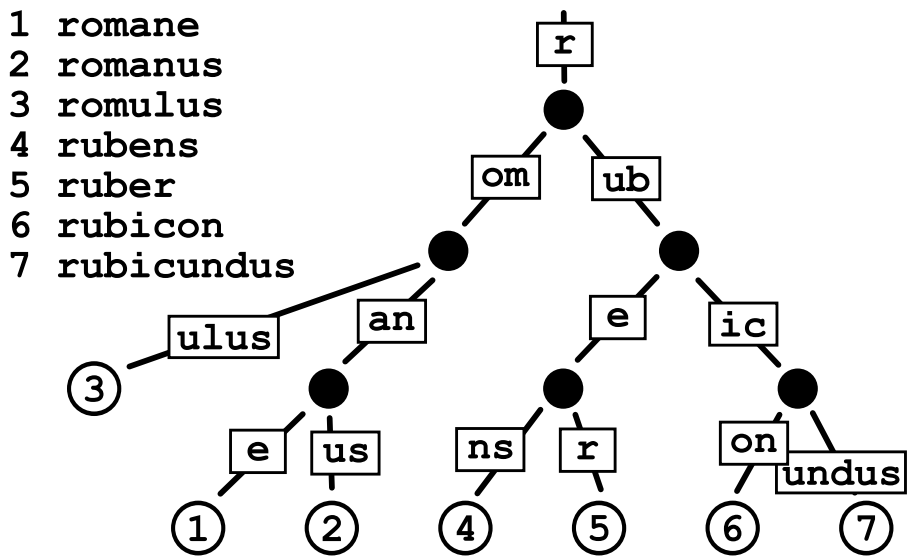
Preslikavanje između adresa i *stanja računa*.

Stanje računa:

- `nonce`: broj transkacija.
- `balance`: iznos kriptovalute.
- `storageRoot`: hash pokazivač na strukturu podataka u kojoj je pohranjena trajna memorija ugovora.
- `codeHash`: hash pokazivač na bytecode ugovora.

Strukture podataka

- *Recursive length prefix* (RLP) kodiranje za sve strukture (transakcija, stanje jednog računa, zaglavlje bloka, . . .).
- *Merkle-Patricia* stabla (MPT) za preslikavanja (globalno stanje, lista transakcija, trajna memorija ugovora, . . .).



Patricia (radix) stablo, Izvor: `wikipedia.org`

Potrošnja goriva – više detalja

- Na početku izvođenja ugovora `gasPrice * gasLimit` se skine s računa pošiljatelja transakcije.
- Svaka operacija prilikom izvođenja potroši neku količinu goriva.
- Ako na kraju izvođenja ostane `gasRem` goriva, onda se pošiljatelju transakcije vraća `gasPrice * gasRem`.
- Ostatak je naknada za rudara.
- Ako nestane goriva tijekom izvršavanja, stanje ugovora se vraća na početno, ali se transakcija smatra ispravnom i naknada ostaje rudaru.
- Kada ugovor zove drugi ugovor može mu ograničiti količinu goriva koju smije potrošiti.

Izvod iz cjenika:

$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	50	Partial payment when multiplied by $d\log_{256}(\text{exponent})e$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the Homestead transition.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Izvor: `ethereum.github.io/yellowpaper`

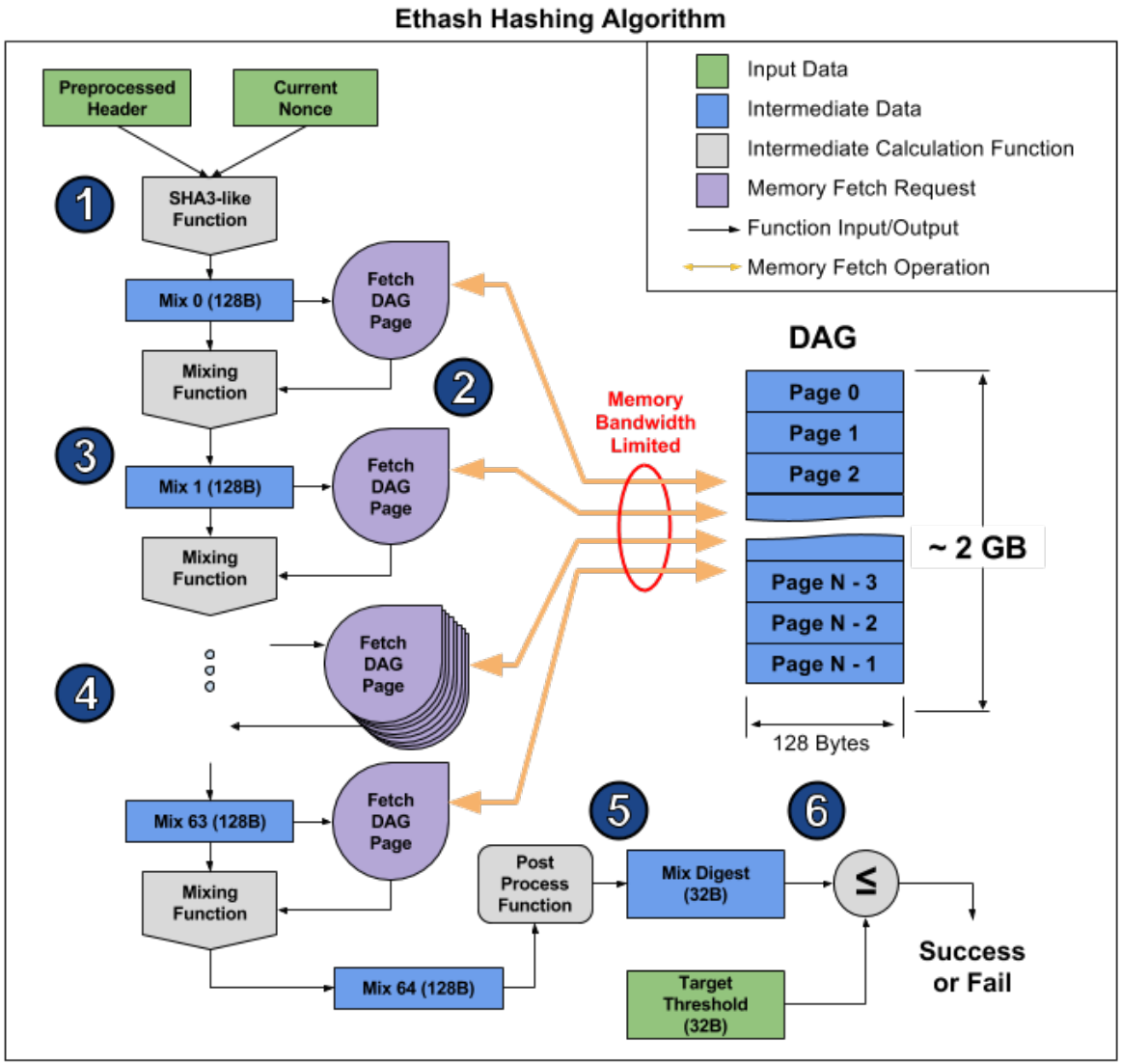
Zaglavlje bloka (odabrana polja)

- `number`: visina bloka.
- `parentHash`: hash pokazivač na prethodni blok.
- `beneficiary`: adresa za nagradu.
- `transactionsRoot`: hash pokazivač na MPT transakcija.
- `receiptsRoot`: hash pokazivač na MPT potvrda.
- `stateRoot`: hash pokazivač na MPT globalnog stanja.
- `gasLimit`: trenutna najveća dozvoljena količina po bloku.
- `gasUsed`: ukupna količina potrošenog goriva.
- `difficulty`: težina rudarenja.
- `mixHash`: parametar za *proof-of-work* funkciju.
- `nonce`: parametar za *proof-of-work* funkciju.

Ciljevi dizajna

Otporno na ASIC.

- Zasićenje memorijske propusnosti (*memory access saturation*).
- Mogućnost isplativog rudarenja pomoću GPU-a.
- Provjera ispravnosti moguća laganim klijentima.
- Rudarenje neisplativo laganim klijentima.



Izvor: nepoznat

Ethash algoritam

- Na temelju broja bloka se generira skup podataka (DAG).
  - DAG se mijenja svakih 3000 blokova.
  - Trenutna veličina (prosinac 2021.) 4.5 GiB.
- Prilikom računanja *ethash* funkcije dohvaćaju se podaci iz DAG-a i hashiraju zajedno s podacima iz zaglavlja bloka.
- Dohvaćanje tih podataka je usko grlo računanja ethash-a.

Lagani klijenti

- Lagani klijent može napraviti verifikaciju bez DAG-a.
- Verifikacija bez DAG-a je bitno sporija, a rudarenje neisplativo.

Željeno vrijeme između dva bloka: 12 sekundi

Koji su problemi s kratkim vremenom između blokova?

Ethereum rješenje: nagrada za *ommmer* blokove (stričeve i strine)

- *Ommer* – ispravan blok koji nije na konsenzus lancu.
- Svaki blok može u zaglavlje uključiti hash pokazivače na dva *ommer* bloka.
- Transakcije u *ommer* blokovima se ignoriraju.
- Rudar *ommer* bloka dobiva 7/8 nagrade za blok.
- Rudar koji je uključio *ommer* blok dobiva 1/32 nagrade za blok.

Trenutno

Protokol baziran na *proof-of-work* konceptu.

- Konsenzus se postiže oko lanca s najvećom ukupnom težinom.

Budućnost

U tijeku je prelazak na *proof-of-stake* koncept. Protokoli kandidati:

- Casper the Friendly Finality Gadget
- Casper the Friendly GHOST: Correct-by-Construction

Faza 0: *Beacon chain* (2020-2021)

Proof-of-stake blockchain koji trenutno radi paralelno Ethereum lancu i samo potvrđuje Ethereum blokove.

Faza 1: *The merge* (2022)

Prelazak Ethereum lanca na proof-of-stake.

Faza 2: *Shard Chains* (2023)

Razdvajanje u 64 lanca koji se održavaju paralelno i sinkroniziraju pomoću *Beacon* lanca.